LA-UR 98-2250

Dynamic Traffic Assignment on Parallel Computers

K. Nagel, R. Frye, R. Jakob, M. Rickert, P. Stretz

Submitted to Proceedings of ISCOPE-98

Dynamic Traffic Assignment

on Parallel Computers

K. Nagel R. Frye R. Jakob M. Rickert P. Stretz

Los Alamos National Laboratory, TSA-DO/SA, MS M997

P.O.Box 1663, Los Alamos NM 87545, USA

Contact: M. Rickert, phone: (505) 665-7285, fax: (505) 665-7464, e-mail: rickert@lanl.gov

May 29, 1998

Abstract

We describe part of the current framework of the TRANSIMS traffic research project at the Los Alamos National Laboratory. It includes parallel implementations of a route planner and a microscopic traffic simulation model. We present performance figures and results of an offline load-balancing scheme used in one of the iterative re-planning runs required for dynamic route assignment.

1 Introduction

The traditional urban transportation planning process consists of four steps: trip generation, trip distribution, modal split, and assignment of routes onto the network.

Iterated microsimulations of traffic provide an alternative for the assignment portion. Several groups (e.g. [1, 4, 6, 7, 5, 15]) have used the iterative approach of *routing – microsimulation –*

feedback of travel-times to obtain an assignment (route set) that is, within the accuracy permitted by any implicit or explicit stochasticity of the model, self-consistent. In this paper we outline part of the framework of TRANSIMS [2, 3] as it was used for an extensive case-study of the Dallas / Fort Worth street network. It consists of two main applications: (a) a route planner based upon a fast implementation of the Dijkstra algorithm that uses time-dependent link travel-times to compute shortest routes, and (b) a traffic microsimulation that executes the routes generated by the planner and supplies a feedback file which is used in subsequent calls of the router.

In the following sections we will describe the two applications and their mutual dependency in more detail.

2 Application Framework

The basic input of the dynamic assignment process is a list of trips[1] defining what vehicles depart from which *origin* at what *departure time* to what *destination*. For all considerations presented in this paper, the list of trips is regarded as given and fixed. The main goal lies in assigning actual route plans to these trips which fulfill a certain optimization criterion, e.g. minimizing each individual's travel-time based upon the actual time-dependent link travel-times which would be generated by executing the route-plans. In TRANSIMS a microsimulation is used to provide (and verify) the link travel-times generated by the route-plan.

Since the first route-set is usually based upon unrealistic travel-times (e.g. vehicles traveling at speed limit), the process of routing of all (or a fraction f_r of all) trips, route execution, and

[1] In general, TRANSIMS uses lists of activities (sleep, eat, work, ...) instead of lists of trips. A trip connects two activities, if they happen at different locations. For the purpose of this paper, talking about trips will be sufficient, though, and simplify notation.

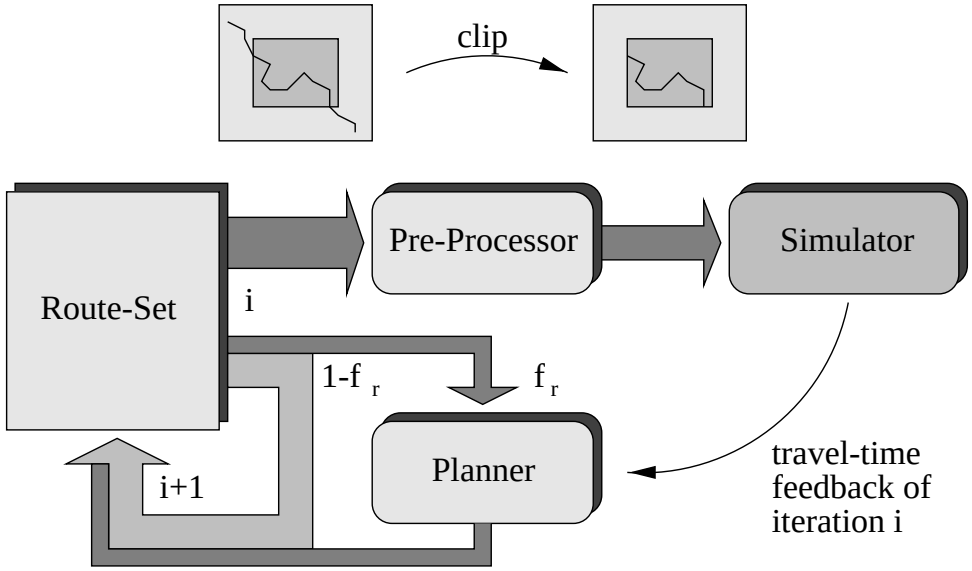

Figure 1: *Iterative assignment with simulation feedback* — For iteration $i+1$ the fraction f_r of the previous route-set is re-planned by the router using link travel-times of iteration i. The remaining fraction $1-f_r$ is simply reused. Before the route-set is fed into the simulation it is clipped to the boundaries of the study-area.

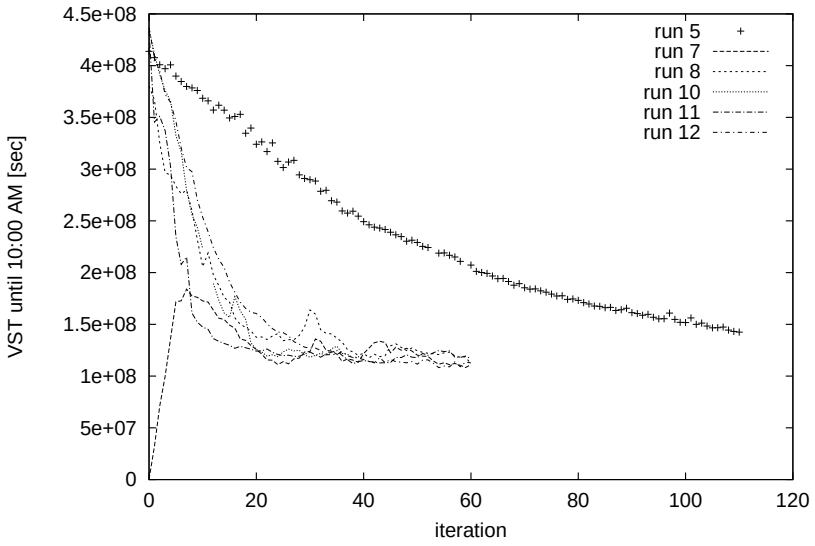

Figure 2: *Relaxation of the accumulated travel-time*

feedback of travel-times has to be repeated in an iterative process. Figure 1 depicts the data flow during an iteration run. It turned out that choosing a re-planning fraction $f_r < 1.0$ improves the relaxation behavior. Figure 2 shows the accumulated travel-time as a function of the iteration number for different iteration strategies and re-planning fractions of 1% and 5% (see [11]). It is obvious that approximately 40 to 60 iterations with 5% re-planning are required before the travel-time has reached a sufficiently relaxed state.

3 Route Planner

During the first iteration the route planner reads the trip file and a (potentially time-dependent) link travel-times file to generate a route set. For each subsequent step, it uses the route set of the previous step (which includes the trips implicitly) and re-routes a fraction of all trips. The travel-time feedback from the microsimulation is averaged over bins of width $t_b = 900$ seconds. Both for the initial route planning and for re-planning, each route is handled independently, i.e. the planner itself does not keep track of the delays that will be caused by travelers on the links they are intending to use.

3.1 Parallelization

The route planner uses a classical master-slave approach with PVM as message passing library. The master reads in the current route-plan and distributes trips by coding them into messages and sending them to the slaves. Each slave reads a copy of the travel-time feedback file for its shortest path computations. The slaves return routes which are sorted with respect to departure time and written to a new route set file by the master. Since the computations of the routes are completely independent, there is no communication between the slaves at all.

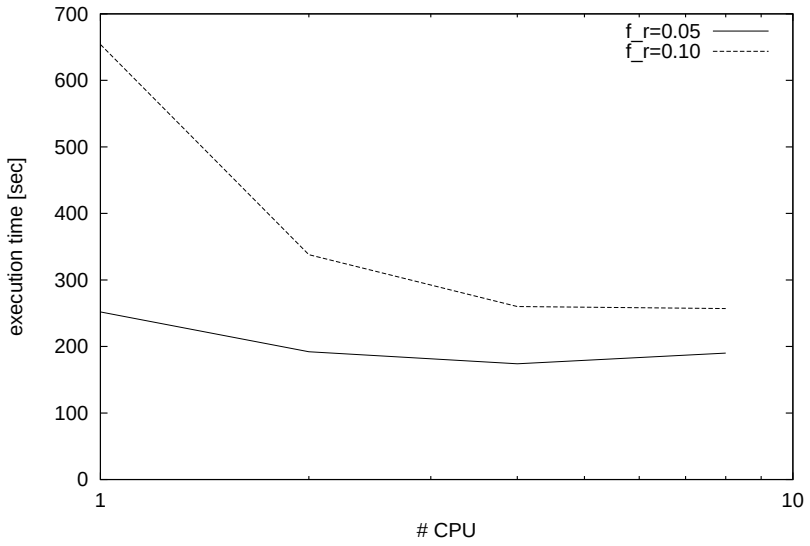

Figure 3: *Execution times of the Route Planner*

Figure 3 shows the execution times for processing 100,000 plans at two different re-planning fractions. The speedup is better for larger re-planning fractions f_r because the ratio between shortest path computations and I/O overhead improves. Still, current speed-up is not very satisfactory and we may have to think about more efficient methods such as packaging several route requests into single messages, or maybe even completely abandoning the master-slave approach.

4 Microsimulation

The cellular automaton (CA) traffic model used in the microsimulation was developed by several authors, first as a single-lane version [10] and later extended to a two-lane version [12]. Its discreteness (both in time and space) as well as its small set of update rules (three for single lane motion and another three for lane changing) permit extremely high execution speeds. Rickert, Wagner and Gawron [13, 14] implemented a parallelized traffic simulation running several times

faster then real–time[2] for the German Autobahn network on an SGI Power Challenger. A comprehensive summary about how the traffic CA has been used in simulation models can be found in [8].

4.1 Parallelization

The inherent structure of a traffic microsimulation favors a *domain composition* as the general approach to parallelization:

- The street network can easily partitioned into tiles of equal or almost equal size. A realistic measure for size is the accumulated length of all streets associated with a tile.
- The range of interdependencies between network elements are restricted to the interaction range of the CA model. All current rule sets have a interaction range of $35[m]$ which is still small with respect to the average link length. Therefore, the network easily separates into independent components.

Therefore, the most straightforward approach is to use an orthogonal recursive bisection for the distribution of the network. Links residing on two CPUs are cut in the middle. As a consequence, the tiles exchange boundary information containing all data necessary for the evaluation of the CA rule sets, resulting only in local communication between neighboring tiles.

The simulation uses a parallel update with a global time-step. However, synchronization of all CPUs is only performed after a *simulation–sequence* comprising approximately 10-20 time–steps. In between, there is only an implicit synchronization through the exchange of local boundaries.

[2]Running at several times real-time means that several simulation seconds can be computed in one wall-clock second.

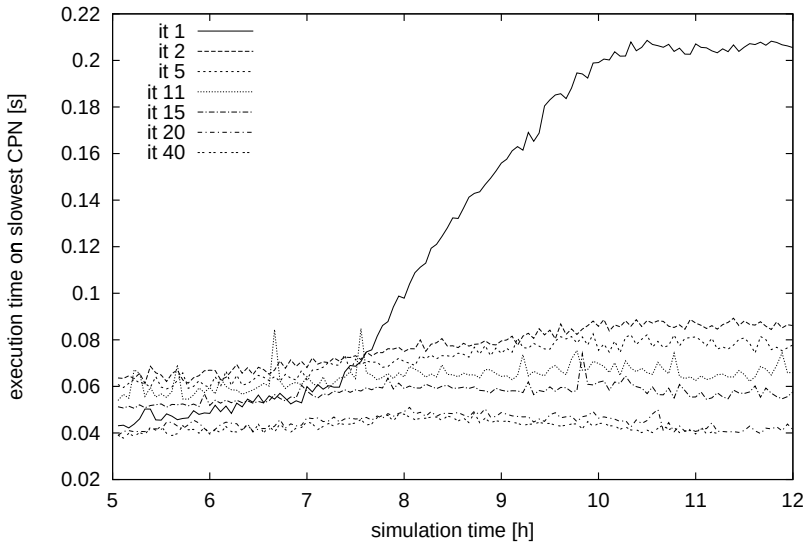

Figure 4: *Execution times with external load feedback*

The global time–step is used to guarantee consistent collection of statistical data: Although partial results from the CPUs may not be collected at the same physical wall-clock time due to a potential time-step gradient (see [9]), they always belong to the same logical time-step. The master CPU takes care of combining partial results.

Each time-step is subdivided into two sub-time-steps. The first sub-time-step is used for lane changing, while the second sub-time-step is used for forward motion. Each sub-time-step requires the exchange of boundaries between CPUs.

The actual implementation of the microsimulation was done by defining descendent C++ classes of the C++ base classes provided in the Parallel Toolbox. The underlying communication library has interfaces for both PVM and MPI. A description of the toolbox is beyond the scope of this paper. More information can be found in [11].

4.2 Off-line Load Balancing

We implemented a simple external feedback for the initial static load balancing. During run time we collect the execution time of each link and each intersection (node). The statistics are output to file every 1000 time-steps. For the next iteration run the file is fed back to the initial load balancing algorithm. In this iteration, instead of using the link lengths as load estimate, the actual execution times are used as distribution criterion.

To verify the impact of this approach we monitored the execution times per time-step throughout the simulation period. Figure 4 depicts the results of one of the iteration runs. For iteration step 1, the load balancer used the link lengths as criterion. The execution times were low until the first grid-locks[3] appear around 7:30 am. The execution time increased fivefold from 0.04 [sec] to 0.2 [sec]. In iteration 2 the execution time is almost independent of the simulation time. Note that due to the equilibration, the execution time for early simulation hours increased from 0.04 [sec] to 0.06 [sec], but this effect is more than compensated later on.

The figure also contains plots for later iterations (11, 15, 20, and 40). The improvement of execution times is mainly due to the route adaptation process: all grid-locks have disappeared and the average vehicle density is much lower.

5 Performance

The trip file contained 300,000 trips occupying approximately 250 MByte of disk space (120 MByte in a slightly compressed binary format). The planning network comprised 9864 nodes and unidirectional 24662 links. The simulation network itself consisted of 2292 nodes and 6124

[3] A grid-lock is a traffic situation in which vehicles get stuck in dense traffic jams which cannot be resolved anymore.

unidirectional links covering an area of about 5x5 miles in the center of the Dallas street network. The original routes generated for the planning network were truncated to the simulation network by a pre-processor. A typical iteration step takes about 6-8 minutes for pre-processing, 30-35 minutes for simulation using eight CPUs (250 MHz) on SUN Enterprise 4000, and 15-20 minutes for re-planning on one CPU.

Starting with the second iteration we use a UNIX pipe to overlay the re-planning and the pre-processing of the next iteration on two CPUs.

6 Summary and Outlook

We presented a framework for dynamic traffic assignment. The fast implementation of the traffic-simulation and the use of parallel computers allowed us to compute a self-consistent route-set within 30 hours. For future computations the speed will have to be increased considerably. Instead of simulating a small part of a network, the next test-bed will comprise all streets of a medium-sized city. This will increase the computational load by a factor of 20. One improvement will consist in increasing the number of CPUs for the computation. Also, the exchange of data between the route planner and the microsimulation should be done in a distributed fashion instead of aggregating and disaggregating all routes in one file for each iteration.

7 Acknowledgements

We thank A. Bachem, R. Schrader and C. Barrett for supporting MR's work as part of the traffic simulation efforts in Cologne ("Verkehrsverbund Verkehrsimulation und Umweltwirkungen NRW") and Los Alamos (TRANSIMS). Computing time on the SGI-1 Challenger of the Regionales

Rechenzentrum Köln and on the workstation cluster at TSA-DO/SA is gratefully acknowledged. The work of MR was supported in part by the "Graduiertenkolleg Scientific Computing Köln".

References

[1] DYNAMIT, see http://its.mit.edu.

[2] TRANSIMS, TRansportation ANalysis and SIMulation System, Los Alamos National Laboratory, Los Alamos, U.S.A. See http://www-transims.tsasa.lanl.gov.

[3] R.J. Beckman et al. TRANSIMS-release 1.0 - the Dallas-Ft. Worth case study. Los Alamos Unclassified Report (LA-UR) 97-4502, Los Alamos National Laboratory, see http://www-transims.tsasa.lanl.gov/research_team/, 1997.

[4] R.H.M. Emmerink, K.W. Axhausen, P. Nijkamp, and P. Rietveld. Effects of information in road transport networks with recurrent congestion. *Transportation*, 22:21, 1995.

[5] C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. In D.E. Wolf and M. Schreckenberg, editors, *Traffic and granular flow II*. Springer, Heidelberg, 1998. Also Report zpr97-307, http://www.zpr.uni-koeln.de.

[6] H.S. Mahmassani, T. Hu, and R. Jayakrishnan. Dynamic traffic assignment and simulation for advanced network informatics (DYNASMART). In N.H. Gartner and G. Improta, editors, *Urban traffic networks: Dynamic flow modeling and control*. Springer, Berlin/New York, 1995.

[7] K. Nagel. Individual adaption in a path-based simulation of the freeway network of Northrhine-Westfalia. *International Journal of Modern Physics C*, 7(6):883, 1996.

[8] K Nagel, M Rickert, and C L Barrett. Large scale traffic simulations. In J. M. L. M. Palma and J. Dongarra, editors, *Vector and Parallel Processing – VECPAR'96*, volume 1215 of *Lecture Notes in Computer Science*, pages 380–402. Springer, 1997.

[9] K. Nagel and A. Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20:125–146, 1994.

[10] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *J. Physique I*, 2:2221, 1992.

[11] M. Rickert. *Traffic simulation on distributed memory computers*. PhD thesis, University of Cologne, Cologne, Germany, 1998.

[12] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A*, 231:534, 1996.

[13] M. Rickert and P. Wagner. Parallel real-time implementation of large-scale, route-plan-driven traffic simulation. *International Journal of Modern Physics C*, 7:133, 1996.

[14] M. Rickert, P. Wagner, and Ch. Gawron. Real-time traffic simulation of the German Autobahn Network. In *Proceedings of the 4th PASA Workshop*. World Scientific, Singapore, 1996.

[15] M. Van Aerde, B. Hellinga, M. Baker, and H. Rakha. INTEGRATION: An overview of traffic simulation features. *Transportation Research Records*, in press.